Pocket Gems NO.1 Strstr()

时间复杂度O(n)，worst case是在111111111112中找11112这种。

第一题，STRSTR，说白了就是String match，看一个String里面有没有一个小的sub string，如果有，返回index。
解法，直接暴力法，最坏情况O(mn)，比如一个是"aaaaaaaaaaaaaaaaaaaab"，一个是"ab"

```
public int strStr(String haystack, String needle) {
        int l1 = haystack.length(), l2 = needle.length();
        if (l1 < l2) return -1;
        if (l2 == 0 || needle == null) return 0;

        int threshold = l1 - l2;
        for (int i = 0; i <= threshold; ++i) {
            if (haystack.substring(i,i+l2).equals(needle)) {
                return i;
            }
        }
        return -1;
    }
```

Pocket Gems NO.2 K Most Frequently Occurring Numbers

Given a unsorted list of N integers, write a program to find k most repeating integers.
follow up是给你sorted number但是是非常长的stream

我用hashmap去计算所有的integer出现的frequency. 再用min heap去得到top k frequency. 最后输出对应的integer O(k + (n-k)logk) = O(nlogk)

这篇文章提供了6种解法 http://www.geeksforgeeks.org/k-l ... ements-in-an-array/

FOLLOW UP: Top k frequent elements in a stream use Max heap-Priority queue in JAVA

- Kth most frequent number

Pocket Gems NO.3 Ternary Expression to Binary Tree

举个例子，就是

a?b?c:d:e转化成

```
       a
     /   \
   b      e
 /   \
c     d
```

这题的基本思路就是用stack，遇到”?"，入栈，遇到":"，出栈。

我用递归写的，time complexity O(nlogn), space O(1). 时间复杂度高是因为要查找String的分割点，这个每次都要O(N)，不知道有么有什么优化?

```
public static TreeNode solve(String s){
        if(s==null || s.length()==0) return null;
        if(s.length()==1) . From 1point 3acres bbs
            return new TreeNode(s.charAt(0));
        int flag=0; int mid=0;
        for(int i=2;i<=s.length()-1;i++){
            if(s.charAt(i)=='?')
                flag++;
            else if(s.charAt(i)==':'){
                if(flag==0){
                    mid=i;
                    break;
                }
                else flag--;
            }
        }
        TreeNode head=new TreeNode(s.charAt(0));
        TreeNode temp_left=solve(s.substring(2,mid));
        TreeNode temp_right=solve(s.substring(mid+1,s.length()));.
        head.left=temp_left;
        head.right=temp_right;
        return head;
    }
```

Pocket Gems NO.4 Lowest Common Ancestor of Binary Tree

如果是Binary Search Tree的话本题很简单，

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(root==null) return root;

        if((p.val<root.val&&q.val>root.val)||(p.val>root.val&&q.val<root.val)){
            return root;
        }
        else if(p.val<root.val&&q.val<root.val){
            return lowestCommonAncestor(root.left,p,q);
        }
        else{
            return lowestCommonAncestor(root.right,p,q);
        }
    }
```

如果是Binary Tree的话，其实也不算复杂，

```
public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
            if(root == null){
                return null;
            }

            if(root == p || root == q){
                return root;
            }

            TreeNode l = lowestCommonAncestor(root.left,p,q);
            TreeNode r = lowestCommonAncestor(root.right,p,q);

            if(l != null && r != null){
                return root;
            }

            return l != null ? l:r;
          }
```

Pocket Gems NO.5 Sort Color

// 搞两个指针，一个指在当前0的最后一个下标，另一个是指在当前1的最后一个下标（2不需要指针因为剩下的都是2了）。
// 进行一次扫描，如果遇到0就两个指针都前进一步并进行赋值，如果遇到1就后一个指针前进一步并赋值

```
public void sortColors(int[] A) {
    if(A==null || A.length==0)
        return;
    int idx0 = 0;
    int idx1 = 0;
    for(int i=0;i<A.length;i++){
        if(A[i]==0){
            A[i] = 2;
            A[idx1++] = 1;
            A[idx0++] = 0;
        }
        else if(A[i]==1){
            A[i] = 2;
            A[idx1++] = 1;
        }
    }
}
```

Pocket Gems NO.6 Inorder Successor in BST

```
class BSTInorderSuccessor{
    // 1）需要parent指针的做法
    // 找inorder successor 分右孩子是否存在的两种情况考虑
    // O(h) h: height of tree
    public static Node inorderSuccessor(Node root, Node node) {
        if (node.right != null) {              // 有右孩子，直接找右子树的最小节点
            return minValue(node.right);
        }
```

```
        // 否则利用父指针不断向上找，直到父节点的值大于当前节点的值
        // 或者该节点成为父节点的右孩子
        Node parent = node.parent;
        //while (parent != null && node == parent.right) {
        while (parent != null && node.data > parent.data) {
            node = parent;
            parent = parent.parent;
        }
        return parent;
    }


// 2）不需要parent指针的做法
// 过程其实就是个从root查找node节点的过程，同时保存旧的比node大的root节点，作为succ
// O(h)
    public static Node inorderSuccessor2(Node root, Node node) {
        if (node.right != null) {               // 有右孩子，直接找右子树的最小节点
            return minValue(node.right);
        }

        Node succ = null;
        while(root != null) {
            if(root.data > node.data) {  // 继续找更小的
                succ = root;             // 后继节点必然比node要大，所以只能在这里保存
                root = root.left;
            }
            else if(root.data < node.data){             // 继续找更大的
                root = root.right;
            }
            else{           // root节点和node节点重复，停止
                break;
            }
        }
        return succ;
    }
```

```
	/*
	 * Given a non-empty binary search tree, return the minimum data value found
	 * in that tree. Note that the entire tree does not need to be searched.
	 */
	public static Node minValue(Node node) {
		Node cur = node;

		// 最小节点必定在最左下角
		while (cur.left != null) {
			cur = cur.left;
		}
		return cur;
	}

}
```

Pocket Gems NO.7 First Occurrence of Binary Search

```
//Algorithm's complexity is O(log n)
	private int firstOccurrenceBinarySearch(int[] source, int needle) {
	   int low = 0;
	   int high = source.length - 1;
	   int firstOccurrence = Integer.MIN_VALUE;

	   while (low <= high) {
	      int middle = low + ((high - low) >>> 1);

	      if (source[middle] == needle) {
	         // key found and we want to search an earlier occurrence
	         firstOccurrence = middle;
	         high = middle - 1;
	      } else if (source[middle] < needle) {
	         low = middle + 1;
	      } else {
	         high = middle - 1;
	      }
	   }

	   if (firstOccurrence != Integer.MIN_VALUE) {
	      return firstOccurrence;
	   }
	   return -(low + 1);  // key not found
	}
```

Max Array
Input : An array of n numbers, and a number k
Output : An array of n numbers
where
output = MAX(input, input[i + 1]..... input[i + k - 1])
example:
[1, 3, 5, 7, 3, 4, 2, 9], k = 3
[5, 7, 7, 7, 4, 9, 9, 9]

用deque

给出一个三角形三个顶点的坐标和一个新的点，问这个点是否在三角形内。

1st method : barycentric coordinate system

Barycentric coordinate allows to express new *p* coordinates as a linear combination of *p1, p2, p3*. More precisely, it defines 3 scalars *a, b, c* such that :

x = a * x1 + b * x2 + c * x3
y = a * y1 + b * y2 + c * y3
a + b + c = 1

The way to compute *a, b, c* is not difficult :

a = ((y2 - y3)*(x - x3) + (x3 - x2)*(y - y3)) / ((y2 - y3)*(x1 - x3) + (x3 - x2)*(y1 - y3))
b = ((y3 - y1)*(x - x3) + (x1 - x3)*(y - y3)) / ((y2 - y3)*(x1 - x3) + (x3 - x2)*(y1 - y3))
c = 1 - a - b

Then we just need to apply the interesting following property :

p lies in *T* if and only if *0* <= *a* <= *1* and *0* <= *b* <= *1* and *0* <= *c* <= *1*

Code sample :

```
function pointInTriangle(x1, y1, x2, y2, x3, y3, x, y:Number):Boolean
{
 var denominator:Number = ((y2 - y3)*(x1 - x3) + (x3 - x2)*(y1 - y3));
 var a:Number = ((y2 - y3)*(x - x3) + (x3 - x2)*(y - y3)) / denominator;
 var b:Number = ((y3 - y1)*(x - x3) + (x1 - x3)*(y - y3)) / denominator;
 var c:Number = 1 - a - b;

 return 0 <= a && a <= 1 && 0 <= b && b <= 1 && 0 <= c && c <= 1;
}
```

3rd method : check sides with dot product

Maybe the most famous method, based on dot product. We assume that *p1, p2, p3* are ordered in counterclockwise. Then we can check if *p* lies at left of the 3 oriented edges *[p1, p2]*, *[p2, p3]* and *[p3, p1].*

For that, first we need to consider the 3 vectors *v1, v2* and *v3* that are respectively left-orthogonal to *[p1, p2]*, *[p2, p3]* and *[p3, p1] :*

v1 = <y2 - y1, -x2 + x1>
v2 = <y3 - y2, -x3 + x2>
v3 = <y1 - y3, -x1 + x3>

Then we get the 3 following vectors :

v1' = <x - x1, y - y1>
v2' = <x - x2, y - y2>
v3' = <x - x3, y - y3>

At last, we compute the 3 dot products :
dot1 = v1 . v1' = (y2 - y1)*(x - x1) + (-x2 + x1)*(y - y1)
dot2 = v1 . v2' = (y3 - y2)*(x - x2) + (-x3 + x2)*(y - y2)
dot3 = v3 . v3' = (y1 - y3)*(x - x3) + (-x1 + x3)*(y - y3)

Finally, we can apply the interesting property :

p lies in *T* if and only if *0* <= *dot1* and *0* <= *dot2* and *0* <= *dot3*

Code sample :

```
function side(x1, y1, x2, y2, x, y:Number):Number
{
 return (y2 - y1)*(x - x1) + (-x2 + x1)*(y - y1);
}

function pointInTriangle(x1, y1, x2, y2, x3, y3, x, y:Number):Boolean
{
 var checkSide1:Boolean = side(x1, y1, x2, y2, x, y) >= 0;
 var checkSide2:Boolean = side(x2, y2, x3, y3, x, y) >= 0;
 var checkSide3:Boolean = side(x3, y3, x1, y1, x, y) >= 0;
 return checkSide1 && checkSide2 && checkSide3;
}
```

word break I

```
public boolean wordBreak(String s, Set<String> wordDict) {
        if(s==null||s.length()==0) return false;
        if(wordDict.size()==0) return false;

        boolean[] f = new boolean[s.length()+1];
        f[0] = true;

        for(int i = 1; i<=s.length();i++){
            for(int j=0;j<i;j++){
                if(f[j]&&wordDict.contains(s.substring(j,i))){
                    f[i]=true;
                    break;
                }
            }
        }
        return f[s.length()];
    }
```

Sliding Window （给你一个数组和一个数k，k是滑动窗口的大小，滑动窗口每次向右移动一个index，输出是一个新的数组，记录每次窗口里的最小值）
example: [1, 2, 5, 10, 3, 4], output should be: [1, 2, 3, 3]

第一轮俩题：
1. Maximum Product Subarray

```
public int maxProduct(int[] nums) {
        if(nums==null||nums.length==0) return 0;
        int max = nums[0];
        int min = nums[0];
        int preMin, preMax = 0;
        int res = nums[0];
        for(int i = 1;i<nums.length;i++){
            preMin = min;
            preMax = max;
            max =
Math.max(Math.max(nums[i],preMax*nums[i]),preMin*nums[i]);
            min =
Math.min(Math.min(nums[i],preMax*nums[i]),preMin*nums[i]);
            res = Math.max(res,max);
        }
        return res;
    }
```

2. Give a String array, return a minimum length String that satisfies the requirement that the relative orders between the chars in the output are consistent with the relative orders in every array element. Assume the output exists.
For example: { "cba", "bd", "ce", "ed" } -> "cbaed" { "gcd", "jd", "fcj" } -> "fgcjd"

第三轮：
（1）clone node graph
（2）given you a string contians "0-9+-*/()", like "(1+3)*4/2"
return the result(integer)

2. 给一个array，找到其中三个index依次增加的数，并且数值也是依次增加。要求O(n) time O(1) extra space。
第二题他看我做不出来曾放宽到O(n) space，虽然最后也没做出来。

Pocket Gems 两轮电面：

记得要有如下的import
```
import java.util.AbstractMap.SimpleEntry;
import java.util.Comparator;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map.Entry;
import java.util.PriorityQueue;
import java.util.Queue;
import java.util.Set;
import java.util.List;
import java.util.ArrayList;
```

第二题，Top kth repeating number。
解法，hash table+PriorityQueue,时间复杂度是O(n*log k);

```
public static List<Integer> kth(int[] arr, int k){
  List<Integer> result = new ArrayList<Integer>();
  if(arr == null || arr.length==0){
    System.out.println("invalid input");
    return result;
  }
  Hashtable<Integer, Integer> table = new Hashtable<Integer, Integer>();
  for(int e : arr){
    int temp = 0;
    if(!table.containsKey(e)){
      temp++;
    }else{
      temp = table.get(e)+1;
    }
    table.put(e, temp);
  }
  if(k > table.size()){
    System.out.println("invalid input");
    return result;
  }

  Comparator<Entry<Integer, Integer>> cmp = new Comparator<Entry<Integer, Integer>>(){
    public int compare(Entry<Integer, Integer> e1, Entry<Integer, Integer> e2){
      return e2.getValue() - e1.getValue();
    }
  };

  Queue<Entry<Integer, Integer>> q = new PriorityQueue<Entry<Integer, Integer>>(k, cmp);
  Set<Integer> ks = table.keySet();
  Iterator<Integer> it = ks.iterator();
  while(it.hasNext()){
    int key = it.next();
    int value = table.get(key);
    Entry<Integer, Integer> en = new SimpleEntry<Integer, Integer>(key, value);

    q.add(en);
  }
  while(k > 0){
    result.add(q.poll().getKey());
    k--;
  }
  return result;
}
```

1. Given unsorted array of int, each element is the length of a rod, return the minimum total cost of combining all rods.
Cost of combining two rods = length of the combined rod.
Example: [3, 4, 6]
Combine 3 4, get rod 7, cost = 7
Combine 7 6, get rod 13, cost = 13

total cost = 7 + 13 = 20
我用MinHeap做的，先把所有的length发到minheap, 然后poll两个出来加起来再放到heap里，同时update total cost。 time complexity O(nlogn)

第一轮： find kth smallest number in a Array

第三轮： 成就系统设计
要求：
1. 界面支持： 成就列表，成就细节，完成情况
2. 成就系统能检测是否玩家完成成就要求， 每个成就之间是独立的（不用考虑是否有先后关系）
3. 系统要求简洁，新的内容可以增加（就是产生新的object），新的类型也可以添加，比如你可以添加一个int coin，新的integer 型。

Max Array
Input : An array of n numbers, and a number k
Output : An array of n numbers
where
output = MAX(input, input[i + 1]..... input[i + k - 1])

example：
[1, 3, 5, 7, 3, 4, 2, 9], k = 3
[5, 7, 7, 7, 4, 9, 9, 9]

类似 Contains duplicate III